

## IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

## KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

## TWÓJ KOSZYK

DODAJ DO KOSZYKA

## CENNIK I INFORMACJE

ZAMÓW INFORMACJE  
O NOWOŚCIACH

ZAMÓW CENNIK

## CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

# Visual Basic. Wzorce projektowe

Autorzy: James W. Cooper

Tłumaczenie: Jaromir Senczyk

Tytuł oryginału: [Visual Basic. Design Patterns. VB 6.0  
and VB.NET](#)

ISBN: 83-7197-710-7

Liczba stron: 440



Ta książka jest praktycznym podręcznikiem tworzenia programów w języku Visual Basic (w wersji 6 oraz .NET) wykorzystujących wzorce projektowe. Może służyć także jako krótkie wprowadzenie do programowania w nowej wersji .NET języka Visual Basic. Wzorce projektowe omówiono w niej w szeregu krótkich rozdziałów, z których każdy przedstawia jeden wzorec i ilustruje jego wykorzystanie za pomocą jednego lub więcej kompletnych i działających programów z okienkowym interfejsem użytkownika. Każdy rozdział zawiera też diagramy UML ilustrujące powiązania pomiędzy klasami występującymi w implementacji wzorca.

Dzięki niniejszej książce czytelnik dowie się, że wzorce projektowe stanowią powszechnie stosowany sposób organizacji obiektów w programach w celu ich uproszczenia i ułatwienia późniejszych modyfikacji. Wzorce projektowe wprowadzają także zasób nowych pojęć, dzięki któremu łatwiej można opisać konstrukcje tworzonych programów.

Programiści w trakcie zapoznawania się z tematyką wzorców projektowych doznają zwykle momentu olśnienia odkrywając potęgę poznawanego wzorca. Moment ten oznacza, że właśnie uświadomili sobie, w jaki sposób mogą wykorzystać dany wzorec w swojej pracy.



# Spis treści

<b>Przedmowa</b> .....	<b>13</b>
<b>Część I Programowanie obiektowe w języku Visual Basic</b> .....	<b>15</b>
<b>Rozdział 1. Czym są wzorce projektowe?</b> .....	<b>17</b>
Definicja wzorców projektowych .....	18
Proces uczenia się .....	20
Studiowanie wzorców projektowych .....	21
Uwagi na temat podejścia obiektowego.....	21
Wzorce projektowe w Visual Basicu .....	22
Jak zorganizowana jest ta książka? .....	22
<b>Rozdział 2. Diagramy UML</b> .....	<b>23</b>
Dziedziczenie .....	24
Interfejsy .....	25
Kompozycja .....	26
Adnotacje .....	27
Diagramy UML — program WithClass.....	27
Pliki projektów Visual Basica.....	28
<b>Rozdział 3. Klasy i obiekty w Visual Basicu</b> .....	<b>29</b>
Prosty program konwersji temperatur.....	29
Tworzenie klasy Temperature .....	30
Konwersja na stopnie Kelvina .....	32
Przesunięcie decyzji do klasy Temperature .....	32
Zastosowanie klas do konwersji wartości i formatu .....	34
Obsługa niedozwolonych wartości .....	36
Klasa String Tokenizer.....	36
Klasy i obiekty .....	38
Zawieranie klas .....	40
Inicjalizacja klasy.....	40
Klasy i metody typu Property .....	41
Kolejny przykład interfejsu klasy — woltomierz .....	43
Klasa vbFile .....	43
Styl programowania w Visual Basicu.....	45
Podsumowanie .....	46
<b>Rozdział 4. Programowanie obiektowe</b> .....	<b>47</b>
Tworzenie obiektów w Visual Basicu .....	48
Tworzenie instancji obiektów .....	49
Program do pomiaru odległości .....	49
Metody wewnątrz obiektów .....	50
Zmienne .....	50

	Przekazywanie parametrów przez referencję i przez wartość.....	51
	Żargon obiektowy .....	51
<b>Rozdział 5.</b>	<b>Tworzenie kontrolki w Visual Basicu.....</b>	<b>53</b>
	Podświetlone pole tekstowe .....	53
	Zmiana wymiarów kontrolki.....	55
	Testowanie kontrolki HiText .....	55
	Właściwości i metody kontrolki .....	56
	Kompilacja .....	57
	Podsumowanie .....	57
<b>Rozdział 6.</b>	<b>Dziedziczenie i interfejsy .....</b>	<b>59</b>
	Interfejsy .....	59
	Symulator inwestycji.....	60
	Implementacja symulatora .....	61
	Kiedy użyć interfejsu? .....	62
	Metody wspólne .....	65
	Interfejsy ukryte .....	66
	Podsumowanie .....	67
<b>Rozdział 7.</b>	<b>Wprowadzenie do języka Visual Basic .NET .....</b>	<b>69</b>
	Różnice w składni Visual Basic .NET .....	69
	Ulepszona składnia funkcji .....	70
	Deklaracje zmiennych i ich zakresy.....	71
	Obiekty w Visual Basic .NET.....	72
	Opcje kompilatora.....	73
	Zmienne numeryczne w Visual Basic .NET.....	74
	Właściwości w języku Visual Basic wersja 6 i VB.NET .....	74
	Skrócona notacja operacji przypisania .....	75
	Języki prekompilowane i zarządzanie pamięcią .....	76
	Klasy w Visual Basic .NET .....	76
	Tworzenie aplikacji w środowisku Visual Basic .NET .....	78
	Najprostszy program okienkowy w Visual Basic .NET .....	79
	Zastosowanie dziedziczenia .....	81
	Konstruktory .....	82
	Programowanie grafiki w Visual Basic .NET.....	84
	Podpowiedzi i kursory .....	85
	Przeciążanie metod .....	85
	Dziedziczenie w języku Visual Basic .NET.....	86
	Przestrzenie nazw.....	87
	Klasa Square.....	88
	Dostęp do składowych klasy .....	89
	Zastępowanie metod w klasach pochodnych .....	89
	Przeciążanie i przesłanianie .....	90
	Zastępowanie kontrolki .....	92
	Interfejsy .....	93
	Podsumowanie .....	94
<b>Rozdział 8.</b>	<b>Tablice, pliki oraz wyjątki w Visual Basic .NET .....</b>	<b>95</b>
	Tablice.....	95
	Kolekcje .....	97
	Klasa ArrayList .....	97
	Tablice z kodowaniem mieszającym .....	98
	Klasa SortedList .....	98
	Wyjątki .....	98
	Obsługa wielu wyjątków .....	99

Wywoływanie wyjątków.....	100
Operacje na plikach.....	100
Klasa File .....	101
Odczyt danych z pliku tekstowego .....	102
Zapis danych w pliku tekstowym.....	102
Wyjątki a operacje na plikach.....	102
Sprawdzanie osiągnięcia końca pliku .....	103
Klasa FileInfo.....	103
Klasa vbFile .....	104
<b>Część II</b>	
<b>Wzorce konstrukcyjne .....</b>	<b>107</b>
<b>Rozdział 9. Wzorec Simple Factory.....</b>	<b>109</b>
Sposób działania wzorca Simple Factory .....	109
Przykładowy program .....	110
Klasy pochodne.....	110
Klasa Simple Factory .....	111
Użycie fabryki .....	112
Implementacja fabryki w Visual Basic .NET .....	113
Fabryki w obliczeniach matematycznych.....	114
Zagadnienia do przemyślenia.....	115
<b>Rozdział 10. Wzorec Factory Method .....</b>	<b>117</b>
Klasa Swimmer .....	120
Klasa Events i jej klasy pochodne.....	120
Rozstawienie bezpośrednie.....	122
Rozstawienie okalające.....	123
Program rozstawiający .....	124
Jeszcze jedno zastosowanie fabryki .....	124
Program rozstawiający w Visual Basic .NET .....	125
Kiedy należy używać wzorca Factory Method? .....	127
Zagadnienia do przemyślenia.....	128
<b>Rozdział 11. Wzorec Abstract Factory .....</b>	<b>129</b>
Abstract Factory i projektowanie ogrodów .....	129
Interfejs użytkownika programu Gardener .....	131
Abstract Factory w Visual Basic .NET .....	133
PictureBox.....	135
Obsługa zdarzeń wyboru.....	135
Rozbudowa programu o kolejne klasy.....	137
Konsekwencje stosowania wzorca Abstract Factory .....	137
Zagadnienia do przemyślenia.....	137
<b>Rozdział 12. Wzorec Singleton .....</b>	<b>139</b>
Wzorec Singleton i metody statyczne .....	139
Obsługa błędów.....	141
Globalny punkt dostępu .....	141
Kontrolka MSComm i wzorec Singleton .....	142
Porty dostępne w systemie .....	143
Wzorec Singleton w Visual Basic .NET .....	144
Zastosowanie konstruktora o dostępie prywatnym .....	145
Obsługa błędów.....	145
Program SpoolDemo.....	146
Globalny punkt dostępu .....	148
Inne konsekwencje wzorca Singleton .....	148
Zagadnienia do przemyślenia.....	148

<b>Rozdział 13. Wzorec Builder.....</b>	<b>149</b>
Program do śledzenia inwestycji.....	150
Zastosowanie wzorca Builder .....	151
Budowniczy okna listy wyboru.....	153
Budowniczy okna pól wyboru .....	154
Implementacja wzorca Builder w Visual Basic .NET .....	155
Klasa StockFactory .....	156
Klasa CheckChoice .....	157
Klasa ListboxChoice .....	158
Kolekcje obiektów klasy Items .....	159
Ostateczna wersja programu .....	160
Konsekwencje zastosowania wzorca Builder .....	160
Zagadnienia do przemyślenia.....	162
<b>Rozdział 14. Wzorec Prototype .....</b>	<b>163</b>
Klonowanie obiektów w Visual Basicu wersja 6.....	164
Zastosowanie prototypu .....	164
Zastosowanie wzorca Prototype.....	167
Dodatkowe metody w klasach pochodnych.....	168
Różne klasy o wspólnym interfejsie .....	169
Menedżer prototypów .....	171
Wzorec Prototype w Visual Basic .NET .....	172
Konsekwencje stosowania wzorca Prototype .....	175
Zagadnienia do przemyślenia.....	176
Podsumowanie wzorców konstrukcyjnych .....	176
<b>Część III Wzorce strukturalne .....</b>	<b>177</b>
<b>Rozdział 15. Wzorec Adapter .....</b>	<b>179</b>
Przenoszenie danych pomiędzy listami .....	179
Zastosowanie MSFlexGrid .....	180
Wykorzystanie TreeView .....	183
Adapter obiektów .....	183
Adaptory w Visual Basic .NET .....	184
Adapter kontrolki TreeView w VisualBasic.NET .....	186
Zastosowanie DataGrid.....	187
Adapter klas .....	188
Adaptory podwójne .....	189
Adaptory obiektów i adaptory klas w Visual Basic .NET.....	190
Adaptory dynamiczne .....	190
Adaptory w języku Visual Basic .....	190
Zagadnienia do przemyślenia.....	190
<b>Rozdział 16. Wzorec Bridge .....</b>	<b>191</b>
Klasy visList .....	194
Diagram klas .....	194
Rozbudowa mostu .....	195
Kontrolki ActiveX jako mosty .....	196
Wzorec Bridge w Visual Basic .NET.....	198
Klasa ProductList .....	198
Klasa ProductTable .....	199
Wczytywanie danych .....	200
Zamiana stron mostu.....	201
Konsekwencje stosowania wzorca Bridge.....	202
Zagadnienia do przemyślenia.....	202

<b>Rozdział 17. Wzorzec Composite .....</b>	<b>203</b>
Implementacja kompozytu .....	204
Obliczanie wynagrodzeń .....	204
Klasa Employee .....	205
Klasa Subords .....	207
Klasa Boss.....	208
Tworzymy drzewo pracowników.....	209
Awans pracownika.....	211
Listy dwukierunkowe.....	212
Konsekwencje stosowania wzorca Composite .....	213
Uproszczony wzorzec Composite.....	213
Kompozyty w języku Visual Basic .....	214
Wzorzec Composite w Visual Basic .NET .....	214
Wyliczenie.....	215
Konstruktory klasy Boss .....	216
Inne zagadnienia implementacji wzorca .....	217
Zagadnienia do przemyślenia.....	217
<b>Rozdział 18. Wzorzec Decorator .....</b>	<b>219</b>
Dekorator CoolButton.....	219
Zastosowanie dekoratora.....	222
Kontrolki ActiveX jako dekoratory .....	225
Dekorator w Visual Basic .NET .....	225
Dekoratory niewizualne .....	227
Dekoratory, adaptery i kompozyty.....	228
Konsekwencje stosowania wzorca Decorator.....	228
Zagadnienia do przemyślenia.....	228
<b>Rozdział 19. Wzorzec Facade .....</b>	<b>229</b>
Czym jest baza danych?.....	229
Uzyskiwanie informacji z bazy danych .....	231
Systemy baz danych.....	231
ODBC.....	232
Połączenia do baz danych w języku Visual Basic .....	232
Struktura dostępu do bazy danych .....	232
Klasa DBase.....	233
Budujemy fasadę.....	235
Klasa Stores.....	237
Tworzymy tabele Stores oraz Foods.....	239
Tabela Price.....	239
Zapytanie o cenę .....	241
Podsumowanie wzorca Facade .....	242
Wykorzystanie interfejsu ADO w języku Visual Basic.....	243
Połączenia ADO.....	243
Przeszukiwanie i dodawanie rekordów .....	244
Wykorzystanie rozszerzeń interfejsu ADO.....	244
Klasa DBase wykorzystująca ADO .....	245
Dostęp do baz danych w Visual Basic .NET.....	248
Wykorzystanie ADO.NET .....	249
Połączenie do bazy danych .....	249
Odczyt danych z bazy .....	249
Wykonanie zapytania .....	250
Usuwanie danych z tabeli.....	250
Dodawanie rekordów do tabeli za pomocą ADO.NET.....	251
Fasada ADO w Visual Basic .NET .....	252
Klasa DBTable .....	253

	Klasy Stores i Foods .....	254
	Klasa Prices.....	256
	Załadowanie danych do bazy.....	257
	Końcowa postać programu.....	259
	W jaki sposób działa fasada? .....	260
	Konsekwencje stosowania wzorca Facade .....	260
	Zagadnienia do przemyślenia.....	260
<b>Rozdział 20.</b>	<b>Wzorzec Flyweight.....</b>	<b>261</b>
	Omówienie .....	262
	Przykład zastosowania wzorca Flyweight.....	262
	Diagram klas .....	265
	Wybór folderu .....	265
	Implementacja wzorca Flyweight w języku Visual Basic .NET.....	267
	Zastosowanie wzorca Flyweight w języku Visual Basic .....	270
	Obiekty współdzielone.....	271
	Obiekty kopiowane podczas zapisu .....	271
	Zagadnienia do przemyślenia.....	271
<b>Rozdział 21.</b>	<b>Wzorzec Proxy.....</b>	<b>273</b>
	Przykładowy program .....	274
	Implementacja wzorca Proxy w języku Visual Basic .NET .....	275
	Wzorzec Proxy w języku Visual Basic .....	277
	Kopiowanie podczas zapisu .....	277
	Porównanie z innymi wzorcami.....	278
	Zagadnienia do przemyślenia.....	278
	Podsumowanie wzorców strukturalnych.....	278
<b>Część IV</b>	<b>Wzorce czynnościowe.....</b>	<b>279</b>
<b>Rozdział 22.</b>	<b>Wzorzec Chain of Responsibility.....</b>	<b>281</b>
	Zastosowania wzorca .....	282
	Przykładowy program .....	282
	Listy .....	285
	Implementacja systemu pomocy.....	287
	Obsługa polecenia pomocy .....	289
	Łącuch czy drzewo?.....	290
	Wzorzec Chain of Responsibility w Visual Basic .NET .....	291
	Rodzaje obsługiwanych żądań .....	293
	Zastosowanie wzorca w języku Visual Basic .....	293
	Konsekwencje stosowania wzorca Chain of Responsibility .....	294
	Zagadnienia do przemyślenia.....	294
<b>Rozdział 23.</b>	<b>Wzorzec Command .....</b>	<b>295</b>
	Motywacja.....	295
	Polecenie jako obiekt .....	296
	Konstrukcja obiektów Command.....	297
	Tablice poleceń .....	298
	Konsekwencje stosowania wzorca Command .....	301
	Implementacja funkcji Undo.....	301
	Wzorzec Command w Visual Basic .NET.....	305
	Interfejs CommandHolder.....	307
	Implementacja funkcji Undo w Visual Basic .NET .....	310
	Wzorzec Command w języku Visual Basic .....	312
	Zagadnienia do przemyślenia.....	312

<b>Rozdział 24. Wzorzec Interpreter .....</b>	<b>313</b>
Motywacja.....	313
Zastosowania.....	313
Przykład prostego raportu .....	314
Interpreter języka .....	315
Obiekty używane podczas parsowania .....	316
Redukcja tokenów do operacji.....	319
Implementacja wzorca Interpreter.....	320
Drzewo składni.....	321
Implementacja wzorca Interpreter w Visual Basicu wersja 6.....	324
Obiekty używane podczas parsowania .....	326
Konsekwencje stosowania wzorca Interpreter .....	327
Zagadnienia do przemyślenia.....	328
<b>Rozdział 25. Wzorzec Iterator.....</b>	<b>329</b>
Motywacja.....	329
Przykład iteratora w Visual Basicu wersja 6 .....	330
Pobranie iteratora .....	331
Iteratory filtrujące .....	332
Wyliczenie filtrowane .....	332
Iteratory w Visual Basic .NET .....	333
Konsekwencje stosowania wzorca Iterator.....	335
<b>Rozdział 26. Wzorzec Mediator.....</b>	<b>337</b>
Przykładowy system .....	337
Interakcje pomiędzy elementami interfejsu .....	338
Przykładowy program .....	339
Inicjalizacja systemu .....	342
Mediatory i obiekty poleceń .....	342
Wzorzec Mediator w języku Visual Basic .NET .....	342
Inicjalizacja .....	344
Obsługa zdarzeń dla nowych kontrolerek .....	345
Konsekwencje stosowania wzorca Mediator .....	346
Mediator z pojedynczym interfejsem.....	347
Kwestie implementacji.....	347
<b>Rozdział 27. Wzorzec Memento.....</b>	<b>349</b>
Motywacja.....	349
Implementacja.....	350
Przykładowy program .....	350
Ostrzeżenie.....	356
Obiekty poleceń interfejsu użytkownika .....	356
Obsługa pozostałych zdarzeń.....	357
Implementacja wzorca Memento w języku Visual Basic .NET .....	358
Konsekwencje stosowania wzorca Memento .....	360
Zagadnienia do przemyślenia.....	360
<b>Rozdział 28. Wzorzec Observer.....</b>	<b>361</b>
Obserwacja zmian kolorów.....	362
Implementacja wzorca Observer w języku Visual Basic .NET .....	364
Inne rodzaje komunikatów.....	366
Konsekwencje stosowania wzorca Observer .....	366
Zagadnienia do przemyślenia.....	367

<b>Rozdział 29. Wzorec State .....</b>	<b>369</b>
Przykładowy program .....	369
Przełączanie stanów .....	373
Interakcja pomiędzy klasami Mediator i StateManager.....	374
Klasa FillState .....	377
Lista wycofań operacji .....	378
Wypełnianie okręgów w Visual Basicu wersja 6.....	380
Implementacja wzorca Pattern w języku Visual Basic .NET .....	381
Mediator — klasa wszechwiedząca? .....	385
Konsekwencje stosowania wzorca State .....	386
Przejścia pomiędzy stanami .....	386
Zagadnienia do przemyślenia.....	386
<b>Rozdział 30. Wzorec Strategy .....</b>	<b>387</b>
Motywacja.....	387
Przykładowy program .....	388
Kontekst .....	389
Polecenia programu.....	390
Strategie dla wykresu liniowego i wykresu słupkowego .....	390
Tworzenie wykresów w języku Visual Basic .....	391
Implementacja wzorca Strategy w języku Visual Basic .NET .....	392
Konsekwencje stosowania wzorca Strategy .....	396
<b>Rozdział 31. Wzorec Template Method.....</b>	<b>397</b>
Motywacja.....	397
Rodzaje metod w klasie bazowej .....	398
Przykładowy program .....	399
Klasa StdTriangle.....	400
Klasa IsoscelesTriangle.....	401
Program rysujący trójkąty.....	402
Szablony i wywołania zwrotne .....	403
Konsekwencje stosowania wzorca Template Method .....	404
<b>Rozdział 32. Wzorec Visitor .....</b>	<b>405</b>
Motywacja.....	405
Zastosowania wzorca Visitor .....	406
Przykładowy program .....	407
Wizytowanie obiektów jednej klasy .....	408
Wizytowanie obiektów wielu klas .....	409
Kierownicy są także pracownikami! .....	410
Uniwersalność wizytatora .....	411
Podwójne wywołania .....	412
Po co to wszystko? .....	412
Wizytowanie serii obiektów.....	413
Implementacja wzorca Visitor w Visual Basicu wersja 6.....	413
Konsekwencje stosowania wzorca Visitor .....	416
Zagadnienia do przemyślenia.....	416

---

<b>Dodatki .....</b>	<b>417</b>
<b>Dodatek A    Przykłady na serwerze FTP .....</b>	<b>419</b>
<b>Bibliografia.....</b>	<b>423</b>
<b>Skorowidz.....</b>	<b>425</b>

## Rozdział 28.

# Wzorzec Observer

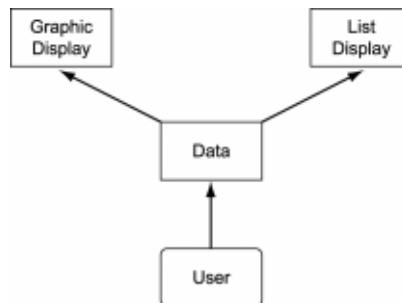
W tym rozdziale przedstawimy, w jaki sposób zastosować wzorzec *Observer* do prezentacji tych samych danych jednocześnie w kilku różnych formach. Jest to częste wymaganie w programach pisanych dla systemów posiadających okienkowy interfejs użytkownika. Na przykład program umożliwiający analizę cen akcji może prezentować dane za pomocą wykresu, tabeli i listy. Każda zmiana cen akcji powinna automatycznie spowodować odpowiednią zmianę we wszystkich formach prezentacji.

Do takiego sposobu działania przyzwyczyli nas popularne programy, jak na przykład arkusz kalkulacyjny *Excel*. Z drugiej strony system Windows nie zawiera żadnego wsparcia umożliwiającego jego łatwą implementację. Gdyby nawet tak było, to i tak programowanie z wykorzystaniem API systemu Windows w języku C jest dość skomplikowane. Tymczasem w programach pisanych w języku Visual Basic możemy osiągnąć pożądany efekt stosując wzorzec *Observer*.

Wzorzec *Observer* zakłada, że osobne obiekty reprezentują dane programu, a osobne zajmują się ich prezentacją, oraz że te drugie *obserwują* zmiany zachodzące w danych. Sytuację tę ilustruje schemat przedstawiony na rysunku 28.1.

**Rysunek 28.1.**

Prezentacja danych za pomocą listy oraz w postaci graficznej



Implementując wzorzec *Observer*, dane określamy zwykle jako *podmiot* (ang. *Subject*), a każdą z form jego prezentacji mianem *obserwatora* (ang. *Observer*). Każdy z *obserwatorów* zgłasza *podmiotowi* zainteresowanie jego danymi poprzez wywołanie metody *podmiotu*. Każdy z *obserwatorów* musi też posiadać znany *podmiotowi* interfejs, którego metody *podmiot* wywołuje, gdy znajdzie zmianę w jego danych. Oba te interfejsy możemy zdefiniować na przykład jak poniżej.

```

'Interfejs Observer
Public Sub sendNotify(msg As String)
End Sub
'-----
'Interface Subject
Public Sub registerInterest(obs As Observer)
End Sub

```

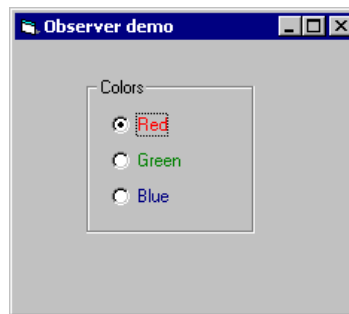
Korzyścią płynącą ze zdefiniowania tych interfejsów jest to, że mogą je łatwo implementować praktycznie dowolne klasy bez względu na ich rzeczywistą funkcjonalność.

## Obserwacja zmian kolorów

Napiszmy zatem prosty program ilustrujący wykorzystanie opisanej koncepcji. Jego interfejs użytkownika zawierać będzie trzy przyciski wyboru, jak przedstawiono to na rysunku 28.2.

### Rysunek 28.2.

*Program  
umożliwiający  
zmianę „danych”  
o kolorach*



Klasa okna programu będzie implementować interfejs Subject. Oznacza to, że musi udostępnić metodę umożliwiającą obserwatorom zgłoszenie zainteresowania przechowywanymi przez nią danymi. Metoda registerInterest będzie umieszczać kolejnego obserwatora w kolekcji.

```

Private Sub Subject_registerInterest(obs As Observer)
    observers.Add obs
End Sub

```

W programie utworzymy dwa obiekty reprezentujące obserwatorów. Jeden będzie pokazywał wybrany kolor i jego nazwę, a drugi doda kolor do listy.

```

Private Sub Form_Load()
    Set observers = New Collection
    'Utwórz obserwatora listy
    Dim lso As New lObserver
    lso.init Me
    lso.Show
    Dim cfr As New ColorForm
    cfr.init Me
    cfr.Show
End Sub

```

Tworząc okno `ColorWindow` zgłosimy jego zainteresowanie danymi głównego okna programu.

```
'Klasa ColorFrame
Implements Observer
Public Sub init(s As Subject)
    s.registerInterest Me
End Sub

Private Sub Observer_sendNotify(msg As String)
    Pic.Cls
    Select Case LCase(msg)
        Case "red"
            Pic.BackColor = vbRed
        Case "green"
            Pic.BackColor = vbGreen
        Case "blue"
            Pic.BackColor = vbBlue
    End Select
    Pic.PSet (300, 600)
    Pic.Print msg
End Sub
```

Okno prezentujące listę wybranych dotąd kolorów będzie także pełnił rolę obserwatora. Reprezentująca je klasa pokazana jest poniżej.

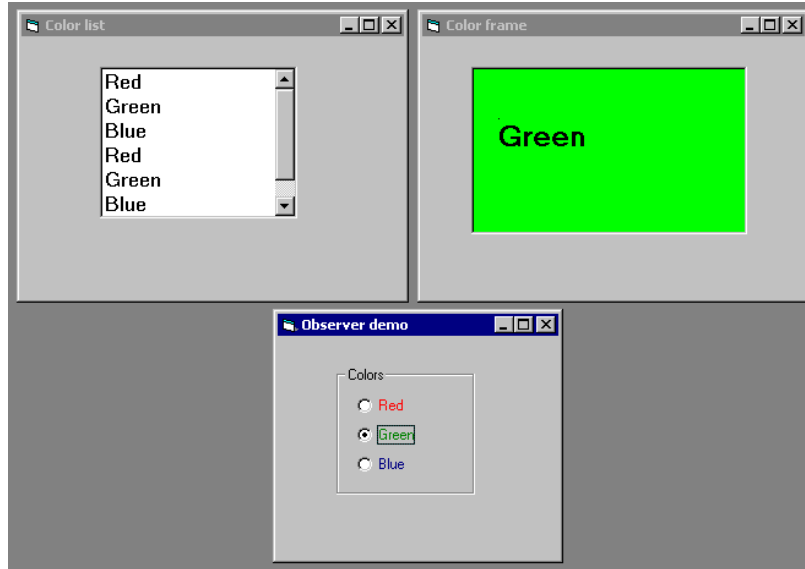
```
'Klasa ListObserver
Implements Observer
Public Sub init(s As Subject)
    s.registerInterest Me
End Sub
'-----
Private Sub Observer_sendNotify(msg As String)
    'Umieść nazwy kolorów na liście
    lsColors.AddItem msg
End Sub
```

Wybór jednego z kolorów za pomocą przycisku w głównym oknie programu spowoduje wywołanie metody `sendNotify` dla każdego zarejestrowanego obserwatora.

```
Private Sub btColor_Click(Index As Integer)
    Dim i As Integer
    Dim msg As String
    Dim obs As Observer
    msg = btColor(Index).Caption 'Pobierz etykietę przycisku
    'Wyślij ją wszystkim obserwatorom
    For i = 1 To observers.Count
        Set obs = observers(i)
        obs.sendNotify msg
    Next i
End Sub
```

W przypadku okna `ColorForm` spowoduje to zmianę koloru wypełniającego obszar kontrolki typu `PictureBox`, a okno `ListForm` doda nazwę wybranego koloru do swojej listy. Działanie programu pokazano na rysunku 28.3.

**Rysunek 28.3.**  
Wybór koloru  
w głównym oknie  
programu  
prezentowany  
jednocześnie  
w formie  
graficznej  
i na liście. Efekt  
zastosowania  
wzorca Observer



## Implementacja wzorca Observer w języku Visual Basic .NET

Podobnie jak w poprzedniej wersji programu, zdefiniujemy najpierw oba interfejsy wzorca *Observer*.

```
Public Interface Observer
    Sub sendNotify(ByVal msg As String)
End Interface
'-----
Public Interface Subject
    Sub registerInterest(ByVal obs As observer)
End Interface
```

Główne okno programu również będzie zawierać trzy przyciski wyboru i reprezentować podmiot zawiadamiający obserwatorów o zmianach. Aby uprościć program, wszystkie trzy przyciski otrzymają wspólną metodę obsługi zdarzeń.

```
Dim evh As EventHandler = New EventHandler(AddressOf radioHandler)
AddHandler opRed.Click, evh
AddHandler opBlue.Click, evh
AddHandler opGreen.Click, evh
```

Metoda ta przesyła obserwatorom łańcuch znaków opisujący wybrany przycisk.

```
Protected Sub RadioHandler(ByVal sender As Object, ByVal e As EventArgs)
    Dim i As Integer
    Dim rbut As RadioButton = CType(sender, RadioButton)
    For i = 0 To observers.Count - 1
```

```

    Dim obs As Observer = CType(observers(i), observer)
    obs.sendNotify(rbut.Text)
Next i
End Sub

```

Klasa reprezentująca obserwatora dodającego nazwę koloru do listy jest praktycznie taka sama jak w poprzedniej wersji programu.

```

Public Class listObs
    Inherits System.Windows.Forms.Form
    Implements Observer
    Public Sub New(ByVal subj As Subject)
        MyBase.New()
        listObs = Me
        InitializeComponent()
        subj.registerInterest(Me)
    End Sub
    '-----
    Public Sub sendNotify(ByVal msg As System.String) Implements Observer.sendNotify
        colors.Items.Add(msg)
    End Sub
End Class

```

Natomiast klasa reprezentująca obserwatora wypełniającego swoje okno kolorem będzie nieco inna. Tekst prezentujący nazwę wybranego koloru pokażemy bezpośrednio w kodzie metody obsługi zdarzenia odrysowania, kolor tła zmienimy w kodzie metody sendNotify.

```

Public Class ColForm
    Inherits System.Windows.Forms.Form
    Implements Observer
    Private colname As String
    Dim fnt As Font
    Dim bBrush As SolidBrush
    '-----
    Public Sub New(ByVal subj As Subject)
        MyBase.New()
        subj.registerInterest(Me)
        ColFrame = Me
        InitializeComponent()
        fnt = New Font("arial", 18, Drawing.FontStyle.Bold)
        bbrush = New SolidBrush(Color.Black)
        AddHandler pic.Paint, New PaintEventHandler(AddressOf painthandler)
    End Sub
    '-----
    Public Sub sendNotify(ByVal msg As System.String) _
        Implements VNetObserver.Observer.sendNotify
        colname = msg
        Select Case msg.ToLower
            Case "red"
                pic.BackColor = color.Red
            Case "blue"
                pic.BackColor = color.Blue
            Case "green"
                pic.BackColor = color.Green
        End Select
    End Sub
    '-----

```

```

Private Sub paintHandler(ByVal sender As Object, ByVal e As PaintEventArgs)
    Dim g As Graphics = e.Graphics
    g.DrawString(colname, fnt, bbrush, 20, 40)
End Sub
End Class

```

## Inne rodzaje komunikatów

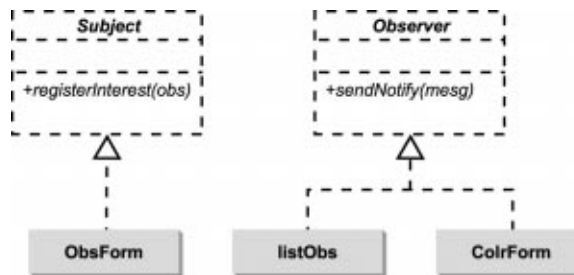
Zastanówmy się teraz, jakie rodzaje komunikatów może wysłać podmiot do obserwatorów? W naszym przykładowym programie rolę komunikatu spełniała etykieta przycisku zawierająca nazwę koloru. Nie zawsze jednak obserwator będzie oczekiwał komunikatu w postaci łańcucha znaków. Łatwo też zauważyć, że w naszym przykładzie dokonujemy dla komunikatu dwu prostych konwersji danych:

1. Pobieramy etykietę przycisku i wysyłamy ją jako ciąg znaków do obserwatora.
2. Obserwator `ColorFrame` zamienia ciąg znaków na odpowiedni kolor.

W bardziej złożonych programach obserwatorzy mogą wymagać różnych, często złożonych komunikatów. Każdy z nich może dokonywać wtedy niezbędnej konwersji, ale w tym celu lepiej będzie zastosować wzorec *Adapter*.

Innym problemem związanym z zastosowaniem wzorca *Observer* będzie obserwacja podmiotu, którego dane mogą zmieniać się w różny sposób. Na przykład pozycje listy mogą być dodawane lub usuwane, ale mogą być też modyfikowane. W takim przypadku musimy wysłać do obserwatorów różne rodzaje komunikatów. Można też wysłać nadal jeden rodzaj komunikatu, a w odpowiedzi obserwator zapyta o charakter zaistniałych zmian.

**Rysunek 28.4.**  
Klasy implementujące interfejsy obserwatora i podmiotu w przykładowym programie wykorzystującym wzorec *Observer*



## Konsekwencje stosowania wzorca Observer

Wzorec *Observer* wprowadza abstrakcyjne powiązania z podmiotem. Podmiot nie zna szczegółów działania żadnego z obserwatorów. Może więc się okazać, że wobec wystąpienia szeregu przyrostowych zmian danych podmiotu zostanie wysłana do obserwatora

seria powtarzających się komunikatów, których obsługa wiązać się będzie ze zbyt dużym kosztem. Rozwiązaniem problemu będzie oczywiście wprowadzenie pewnej dodatkowej logiki, tak by informacje o zmianach nie były wysyłane zbyt wcześnie lub zbyt często.

Inny problem występuje w przypadku, gdy zmiana danych podmiotu dokonywana jest przez pewne części kodu lub systemu zwane dalej klientami. Pojawia się wtedy pytanie, kto powinien inicjować wysłanie komunikatu o zmianach. Jeśli odpowiedzialny będzie za to, jak dotychczas, sam podmiot, to w przypadku wykonywania zmian przez kilku klientów znowu mogą pojawić się serie komunikatów o nieznacznych w istocie zmianach. Można ich uniknąć, jeśli to klient będzie informował podmiot, że należy wysłać komunikat. Jeśli jednak któryś z klientów „zapomni” o poinformowaniu podmiotu, to program nie będzie już działał zgodnie z oczekiwaniami.

Stosując wzorzec *Observer* można także zdefiniować kilka rodzajów komunikatów. W tym celu interfejs obserwatora może definiować kilka różnych metod powiadomienia. Dzięki temu w pewnych sytuacjach obserwator będzie mógł ignorować niektóre z nich.

## Zagadnienia do przemyślenia

Wersja przykładowego programu, którą napisaliśmy korzystając z wersji 6 języka Visual Basic różni się w działaniu od wersji napisanej w Visual Basic .NET tym, że zamknięcie któregokolwiek z okien obserwatorów nie powoduje zamknięcia pozostałych okien i zakończenia pracy programu. W jaki sposób można zastosować dodatkowego obserwatora, aby uzyskać efekt w postaci zakończenia programu?